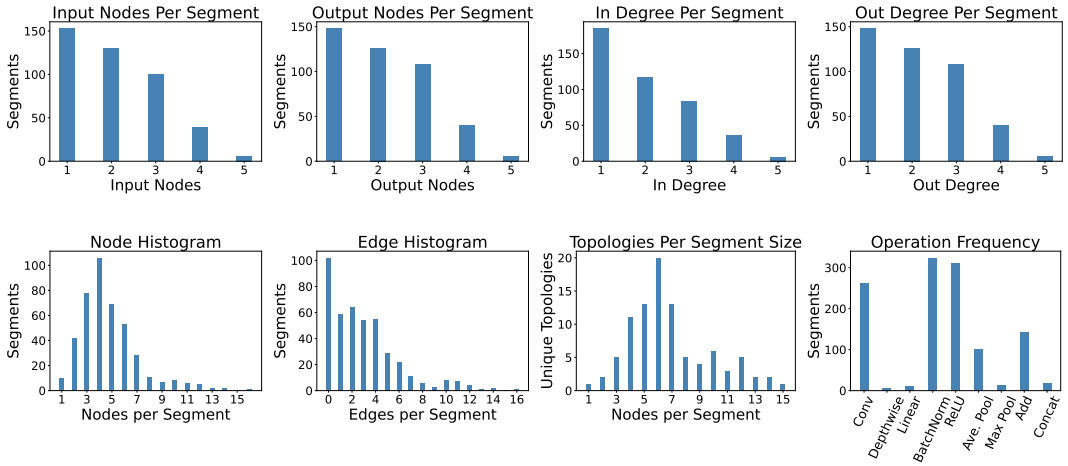


Figure 5: Histograms of segment database statistics including number of input and output nodes/degrees, nodes/edges per segment, unique segment topologies and operation frequency.

A Supplementary Material

A.1 Additional Database Statistics

Figure 5 provides histograms regarding our segment database. Additionally, we enumerate the primitive operations that are only present in specific NAS-Benchmark families:

- **Depthwise**: Inception.
- **Max Pool**: NB-101, Inception and Two-Path.
- **Concat**: NB-101, Inception and Two-Path.

All other operation primitives, e.g., Conv, ReLU, BatchNorm, etc., are present across all 5 CIFAR-10 NAS-Benchmarks.

A.2 Predictor and Dataset Details

We further elaborate on the baseline GNN and PSC predictors from Section 4.1. We provide implementation details, dataset statistics and data pre-processing techniques. We train our predictors for 40 epochs with a batch size of 32 and an initial learning rate of $1e^{-4}$.

A.2.1 Baseline and PSC Predictor Setup

We use the same baseline GNN predictor as GENNAPE [43]. First, CGs are given as input into an initial set of embedding layers that transform discrete node features, such as operation type, input/output tensor resolution, kernel size, and bias, into a continuous vector. The node embeddings are then fed through a series of 6 k-GNN [46] layers. Next, an overall graph embedding is computed by taking the mean of all node embeddings. A simple MLP with 4 hidden layers uses graph embedding to predict performance.

The PSC predictor differs in that each CG sample is first split into its respective *Predecessor*, *Segment*, and *suCcessor* subgraphs before being fed into the predictor. All three subgraphs are processed as separate CGs by the node embedding and k-GNN layers to produce three distinct graph embeddings. We concatenate these graph embeddings feature-wise and feed them into an MLP to generate a prediction. Also, the weights of the node embedding and k-GNN layers are shared for each subgraph type.

A.2.2 Dataset Statistics and PSC Preprocessing

Table 7: Number of Computation Graphs (CG), segment samples and test SRCC folds for each family. We randomly sample 5k NB-101 architectures and only consider NB-201 networks that do not have the 'none' operation.

Arch. Family	CGs	Segments	Folds
NB-101	5.0k	404.9k	42
NB-201	4096	252.8k	34
HiAML	4.6k	65.1k	10
Inception	580	222.4k	129
Two-Path	6.9k	193.1k	10

We train and evaluate the baseline GNN predictor on every unique CG sample. Additional steps are required to train the PSC predictor since each CG comprises many segments and can decompose into many distinct $\{P, S, C\}$ subgraph sets.

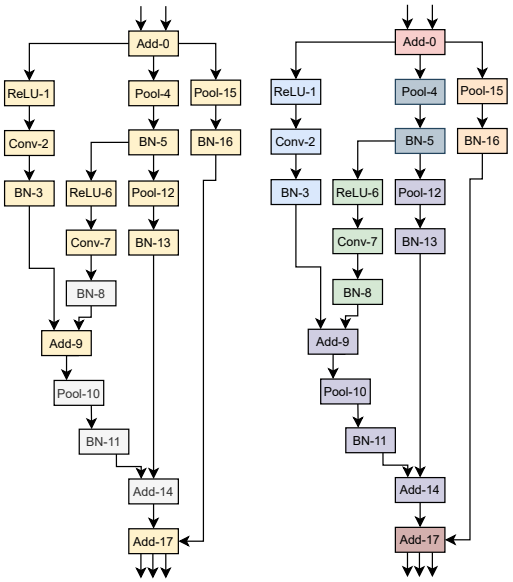
For the intermediate baseline, **PSC 1:1 Ratio** in Table 1, we randomly sample 1 $\{P, S, C\}$ representation from each segmented CG in our training dataset. Hence, the number of samples equals the original number of training instances. For the full **PSC** predictor, we remove this restriction and consider all possible $\{P, S, C\}$ decompositions which drastically increases the number of samples.

Table 7 lists the number of CGs and $\{P, S, C\}$ samples per family. While each $\{P, S, C\}$ sample for a given CG focuses on a different network segment, they still describe the same overall architecture and thus retain the same accuracy label. Therefore, when measuring test SRCC on the PSC predictor, we divide the test data into *folds*. Each fold contains only one $\{P, S, C\}$ instance of a given CG. This avoids introducing additional ties in the ground-truth labels when calculating SRCC. The number of folds is equal to the minimum number of segments in any test CGs or 10, whichever is smaller. Therefore, we calculate the overall test SRCC by gauging SRCC across each fold and averaging the results.

A.3 Segment Extraction with BPE

We compare our BPE subgraph extraction approach to the Weisfeiler-Leman (WL) Kernel method adopted by NAS-BOWL [51] in terms of efficiency. NAS-BOWL applied it on the original, shallow cell-based network representation of NAS-Bench-201 with a depth of 2. We use the WL-kernel on the CG-level and enumerate all subgraphs with a maximum depth of 5. The time and RAM costs of using the WL-kernel scale poorly as we increase the number of graphs and nodes per graph. For example, it takes at least 6 hours to extract and count subgraphs from each NAS-Benchmark family. Moreover, we could not use more than 1k CGs from the HiAML or NB-201 families ($\sim$110 and $\sim$250 nodes per CG, respectively) without facing memory issues on the rack server described in Section A.9.

By contrast, our approach brings several benefits over mining on large graphs with WL-kernels. The extraction process on sequences is efficient. Using BPE enables segment extraction from all benchmark families (over 21k CGs per Tab. 7) simultaneously in less than 20 minutes using around 10GB of RAM. Also, BPE provides segments that are easier to mutate and alleviates limitations with WL-kernel extraction process by topologically ordering the nodes. Figure 6 compares WL and BPE segmentations on a part of a CG from the NAS-Bench-201 family. The subgraph extracted from the WL method (Fig. 6(a)) cannot cover several nodes within its context (grey nodes of BN-8, Pool-10, BN-11, and Add-14) due to a limited depth of 5 and several resid-



(a) WL-kernel (b) BPE

Figure 6: Comparison between subgraphs extracted with WL-kernel and BPE on a NAS-Bench-201 cell. Nodes are numerically labeled by a topological ordering. Best viewed in color. Specifically, WL-kernel extracts one large subgraph consisting of all highlighted nodes (greyed-out nodes are omitted). For BPE, all nodes are extracted into one subgraph, denoted by a unique color.

ual connections. This exacerbates the mutation process. In contrast, segmentation with BPE (Fig. 6(b)) spans different subgraph sizes denoted by separate colors.

A.4 Architecture Training Hyperparameters

We elaborate on the training recipes we use to evaluate input baseline architectures as well as those found by AutoGO.

A.4.1 CIFAR-10 Families

We use the CG representation of the initial and mutated architectures to instantiate networks and train them using TensorFlow. We evaluate CIFAR-10 networks by training them 3 times for 200 epochs with a batch size of 256. We optimize the models using RMSProp with an initial learning rate of $1e^{-3}$ and a momentum factor of 0.9. We anneal the learning rate according to a cosine schedule.

A.4.2 ImageNet, Segmentation and Pose Estimation

When evaluating ResNet and VGG[2] architectures, we first train on ImageNet [53] using timm [62] with a batch size of 1024. We use an initial learning rate of 0.1 which we anneal using a cosine schedule. We optimize the model using Stochastic Gradient Descent (SGD) with a momentum factor of 0.9 and a weight decay of $1e^{-4}$. We set a gradient clipping value of 5.0 and use label smoothing with ϵ = 0.1. We train ResNets for 200 epochs and VGG-16 for 100 epochs. We save the trained weights to fine-tune on other tasks.

We evaluate Semantic Segmentation performance using semseg [70]. The PSPNet [71] head requires two inputs to implement properly. The first is the final latent tensor that originally feeds into the classifier head, while the second requires grafting an auxiliary residual connection $3/4$ths of the way through the network feature extractor. Furthermore, we adjust the dilation factor and strides of all convolution and pooling operations in the later part of the network to limit downsampling. After loading the pretrained ImageNet weights, we fine-tune on Cityscapes [14] images cropped to 713^2 for 200 epochs using a batch size of 16. We use SDG with an initial learning rate of 0.01, a momentum factor of 0.9, and a weight decay of $1e^{-4}$.

We implement 2D Human Pose Estimation using [73]. To convert an ImageNet network, we remove the classifier layers and then append a series of 'Deconvolution-BatchNorm-ReLU' blocks which gradually upsample the latent tensors from 8^2 to 64^2. We train on MPII [4] images cropped to 256^2 for 140 epochs with a batch size of 32. We optimize our networks using Adam, setting an initial learning rate of $1e^{-3}$ for ResNet-50 and VGG-16, and $5e^{-4}$ for ResNet-101. We reduce the learning rate by a factor of 10 at epochs 90 and 120. Finally, we report performance in terms of the Percentage of Correct Keypoints (PCK), specifically the Percentage of Correct Keypoints at a head-neck distance of 0.5 (PCK@h0.5) [72].

A.4.3 Super Resolution

We train networks on DIV2K in the 2x upsampling setting for 1000 epochs with a batch size of 16. We set an input patch size of 64 for EDSR and 48 for FSRCNN. We minimize the L1 loss using the Adam optimizer with an initial learning rate of $1e^{-4}$, which we reduce using a cosine decay schedule.

A.4.4 Image Denoising

We train networks on a custom in-house image-denoising dataset with 7k images. We set an input patch size of 128 for all networks. We train each network for 2k epochs under a batch size of 128. We minimize the L1 loss using the Adam optimizer with an initial learning rate of $1e^{-3}$ and a final learning rate of $1e^{-6}$, reduced over a polynomial schedule.

A.5 CIFAR-10 FLOPs Restraint Ablation

Table 8 provides a full ablation study of AutoGO on all 5 CIFAR-10 families in terms of FLOPs reduction constraint. We consider two settings where AutoGO can reduce FLOPs by at most -20%

[2]Base model uses batch normalization

Table 8: Full ablation study of AutoGO on all 5 CIFAR-10 families considering choice of mutation unit {Operation, Segment}, predictor {GNN, PSC} and FLOPs [1e6] reduction (δ) constraint {-20%, -100%}, extending the results of Table 2. For each experiment, we report the accuracy [%] and FLOPs [1e6] (raw and Δ relative to the baseline). We bold and italicize the best and second best result per family, respectively.

	Baseline		**Operator + GNN**		**Segment + GNN**		**Segment + PSC**	
Family (δFLOPs)	**Acc.**	**FLOPs**	**Acc.**	**FLOPs**	**Acc.**	**FLOPs**	**Acc.**	**FLOPs**
NB-101 (-20%)	95.18%	11722	95.16%	9407	*95.31%*	*10817*	95.06%	9606
Δ			-0.02%	-19.75%	*+0.13%*	*-7.72%*	-0.12%	-18.02%
NB-101 (-100%)			93.12%	1591	95.25%	10513	**95.45%**	**11118**
Δ			-2.06%	-86.42%	+0.07%	-10.31%	**+0.27%**	**-5.15%**
NB-201 (-20%)	93.50%	313	93.28%	250	92.86%	250	93.32%	251
Δ			-0.22%	-20.00%	-0.34%	-20.00%	-0.18%	-19.82%
NB-201 (-100%)			93.37%	232	*93.57%*	*294*	**93.84%**	**303**
Δ			-0.13%	-25.77%	*+0.07%*	*-6.08%*	**+0.34%**	**-3.21%**
HiAML (-20%)	92.32%	246	92.00%	198	92.08%	198	92.22%	230
Δ			-0.32%	-19.60%	-0.24%	-19.76%	-0.10%	-6.82%
HiAML (-100%)			84.63%	28	*92.62%*	*168*	**92.75%**	**198**
Δ			-7.69%	-88.47%	*+0.30%*	*-31.62%*	**+0.43%**	**-19.76%**
Inception (-20%)	93.50%	494	92.97%	399	93.12%	399	**93.52%**	**474**
Δ			-0.23%	-19.23%	-0.08%	-19.28%	**+0.32%**	**-3.96%**
Inception (-100%)			92.97%	319	*93.31%*	*461*	93.30%	478
Δ			-0.23%	-35.35%	*+0.11%*	*-6.72%*	+0.10%	-3.20%
Two-Path (-20%)	87.90%	116	88.63%	106	88.31%	93	88.68%	94
Δ			+0.73%	-8.61%	+0.41%	-20.00%	+0.78%	-19.36%
Two-Path (-100%)			88.63%	106	**89.16%**	**48**	*88.94%*	*91*
Δ			+0.73%	-8.61%	**+1.26%**	**-58.58%**	*+1.04%*	*-21.29%*

relative to the baseline architecture, or can reduce them freely (-100%), while always limiting FLOPs *increases* to be at most +10%. We again note how the best architecture for each family was found using segment mutations.

We observe that the segment-level mutation is a better fit for finding high-performance architectures under wider FLOPs constraints. For example, on HiAML, the segment-mutation cannot improve the accuracy of the base architecture when we impose a FLOPs reduction limit of -20%, yet it can increase the accuracy by up to 0.43% on average when we remove the restriction, even though the best architecture only reduces FLOPs by -19.76%. From this result, we infer that FLOPs restrictions hamper the exploration of the segment-level mutation. The only family where the -20% FLOPs constraint produces a better architecture than the no-constraint setting is Inception, which is already the second-largest family with a base model size of nearly 500 MegaFLOPs. By contrast, the operation-level mutations require FLOPs reduction constraints to break even with the baseline architectures. For example, when no FLOPs constraint is imposed, the operation-level mutation will find HiAML and NB-101 architectures that remove enough convolution nodes to reduce the model size by more than 85%. These changes drastically reduce the accuracy by over 7.5% on HiAML.

A.6 Additional AutoGO Search Details

We provide additional details on the AutoGO search algorithm from Section 3.2.

A.6.1 Node Labeling

Before segmentation with BPE, we label nodes in the CG in the form of *[current operation, incoming operations, outgoing operations]*. We encode each unique node label with a single Chinese character symbol, as they span a wide range of symbols compared to other languages.

A.6.2 Selecting a Sparse BPE Vocabulary

When generating V' as a vocabulary set utilized by BPE to segment CGs, we include all single-node segments as these represent the irreducible primitive operations that must exist within the vocabulary in some form and only filter out multi-node segments.

A.6.3 Selecting Non-Pareto Optimal Architectures

When transitioning from iteration e to $e+1$, we select k architectures from the Pareto frontier $\mathcal{O}$ and search history to serve as parents. If we have sufficient architectures on the Pareto frontier, $|\mathcal{O}| \geq k$, we randomly sample from it. However, if $|\mathcal{O}| < k$, there is an architecture deficit. We compensate for this deficit by selecting non-Pareto optimal architectures that aim to achieve our search objective. We select these architectures by ranking them in terms of predicted accuracy and FLOPs, where higher and lower are better, respectively. We then sum these ranks and select the non-Pareto optimal architectures with the lowest rank sum. Table 9 provides a simple example of this process. Note how the selection mechanism excludes architectures that have high performance but are too large, as well as underperforming architectures.

Table 9: Example of the minimum sum of ranks selection algorithm with a deficit of 3 architectures.

Acc. [%]	Rank	FLOPs	Rank	Rank Sum	Selected?
91.21	0	260	5	5	No
91.10	1	215	2	3	**Yes**
91.02	2	200	0	2	**Yes**
90.75	3	210	1	4	**Yes**
90.35	4	220	3	7	No
89.05	5	250	4	9	No

A.6.4 Segment Selection

For each CG g, we sample a set of m source segments s_i. We sort the segments $\mathcal{S}_g$ by FLOPs and then we select the $m/2$ segments with the lowest FLOPs while randomly sampling the rest.

A.6.5 Accuracy Predictions and FLOPs Constraints

Once we have a set of valid source and replacement segments, we use the PSC predictor to select mutations that yield the most significant accuracy gain. We use a FLOPs calculator (or a proprietary profiling tool for measuring NPU latency/power) to further filter these mutations by rejecting child architectures whose FLOPs deviate too far from the FLOPs of the input architecture.

A.6.6 Resolution Propagation

Adjustment can not always lead to a solution, meaning the replacement segment can not be used for mutation at this position and generate a valid CG. We cast this task as a search problem over the height, width, and channel resolution values on the replacement segment operations. The search spans mutable operations such as convolutions and pooling. The rest of the operations are immutable and only forward the resolution without changing its sizes, such as add, activation functions, and batch normalization. During the search, we limit the adjustments on the values of height, width, and channel sizes to doubling, halving, or keeping the same.

Our solution is based on Mixed Integer Linear Programming (MILP). MILP is an optimization problem formulated with linear objectives, linear constraints, and integer-valued variables. The input to MILP is the replacement segment

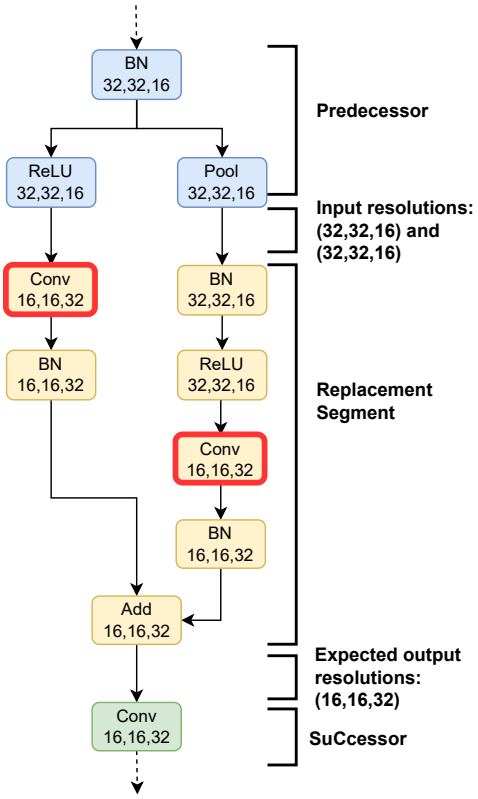


Figure 7: Resolution propagation adjusts the resolution of mutable operations in the replacement segment. The Height, Width, and Channel sizes are adjusted in both 'Conv' operations so that the replacement segment yields the expected output resolution at the 'Add' operation.

DAG. Each node has two variables per each height, width, and channel dimension, denoting input and output resolutions. Each edge is associated with a "flow" variable. We define MILP constraints that regulate the correct flow of resolution. Immutable nodes have input resolutions equal to output resolutions. The output resolution for mutable nodes is less than or equal to the input resolution. The model is optimized to achieve the expected resolution at the output nodes. The model is proven infeasible if the search fails to achieve expected output resolutions.

We briefly illustrate the resolution propagation process. Figure 7 shows a replacement segment (yellow) that is being put together with the Predecessor (blue) and Successor (green) partitions of the network. We provide the output resolution of each operation in the form of (height, width, channel). Notice how the number of input and output nodes of the replacement segment matches the number of output and input nodes of the Predecessor and Successor, respectively. Initially, the replacement segment expects input dimension sizes for its 'Conv' and 'BN' operations of (32, 32, 16), which are the resolutions of the Predecessor's output nodes. Also, the Succesor expects an input size of (16, 16, 32), which demands the replacement segment to output a feature map with this dimension at the 'Add' operation. This requires adjusting the resolution of the 2 mutable 'Conv' operations in the replacement segment (highlighted with red borders). Notice that adjusting one of them or leaving resolutions unadjusted will result in incorrect propagation because the 'Add' operations require its incoming tensors to have the exact same dimensions. We use MILP to solve this problem by finding the correct adjustment to mutable operations by halving, doubling, or maintaining resolution sizes.

A.7 AutoGO Components Evaluation

We evaluate the search efficiency on the benchmark families by measuring the speed of each component. The time to execute the search largely depends on the choice of input architecture, i.e., architectures with more nodes and complex topologies like Inception form large search spaces. On the HiAML and NB-201 families, it takes 15 minutes on average to execute a search iteration using the PSC predictor and segment-level mutation. AutoGO visits over 1000 unique architectures per iteration and can find high-performance architectures in around an hour or less.

Specifically, it takes around 1.5 to 2 minutes to segment a parent architecture using BPE, select source and replacement segments, perform resolution propagation, and rank the mutations using the predictor. The bulk of this time is spent between searching the database for replacement segments, confirming their validity and measuring the performance of each mutation, while the BPE segmentation and source segment selection processes take less than 1 millisecond each. When gauging execution time, we sequentially mutate each parent architecture per iteration, but note that this process can be sped up with parallelization.

Resolution propagation with MILP takes, on average 0.11 seconds to find a solution or determine that the problem is infeasible. We compare it to an exhaustive search approach by enumerating all candidate solutions. It takes, on average, 0.4 seconds to find a solution and more than 4 seconds for infeasible solutions. Our subgraph extraction process for generating the segment vocabulary is very efficient as the BPE operates on a sequence representation of the CGs. It takes less than 20 minutes to sort all CG topologically, and extract subsequences with BPE.

To provide specific examples of the search time, consider the ResNet-50 Arch 2 and EDSR Arch 2 architectures from Tables 3 and 4, respectively. Mutating the initial ResNet-50 and EDSR CGs takes 1.8 and 1.5 minutes, respectively, on our hardware. It takes longer to mutate ResNet-50 simply because the CG contains more nodes (108) than EDSR, whose CG only has 67 nodes. Moreover, since the base EDSR architecture only uses Convolutions and ReLU operations, we exclude segments that contain batchnorm and pooling operations, which reduces the number of replacement segments to consider during mutation.

The first iteration of AutoGO mutates the initial architecture while all subsequent iterations mutate 10 parent architectures. Given that ResNet-50 Arch 2 was found in iteration 3, it took AutoGO around

$$1.8\text{min} + 2\text{iter} * 10\text{arch/iter} * 1.8\text{min/arch} = 37.8\text{min}$$

to discover that architecture. Likewise, EDSR Arch 2 was found in iteration 5, which took

$$1.5\text{min} + 4\text{iter} * 10\text{arch/iter} * 1.5\text{min/arch} = 61.5\text{min}$$

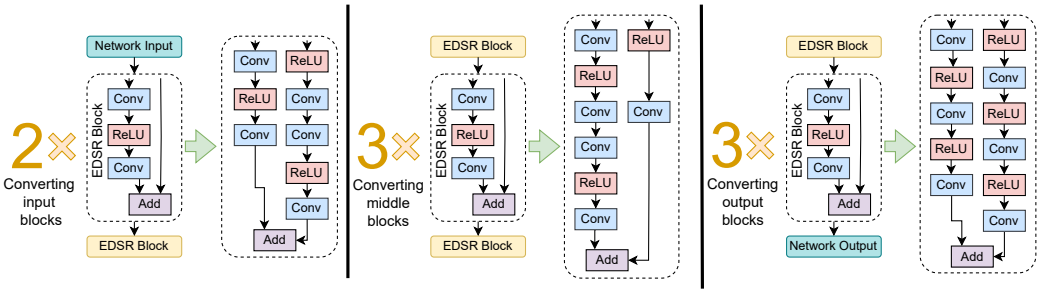


Figure 8: Example mutations performed by AutoGO to create an EDSR mutant by swapping out 8 EDSR blocks. Specifically, AutoGO will swap out multiple, simple 'Conv-ReLU-Conv' residual blocks for larger blocks that have operations on both branches.

to find. Finally, we note that these calculations assume sequential processing of parent architectures. However, it is possible to use multi-processing techniques to mutate multiple parent architectures simultaneously to further speedup the process.

A.8 EDSR Mutation Example

Figure 8 illustrates three distinct mutations that take place to produce an EDSR AutoGO architecture. Initially, the EDSR backbone contains 16 'Conv-ReLU-Conv' residual blocks. To create the mutant network, AutoGO removed 8 of these blocks, denoting half the backbone structure, and replaced them with three double-branch structures that also consist of just convolutions and ReLU activations.

A.9 Hardware and Software Setup

We run our experiments on rack servers using Intel Xeon Gold 6140 CPUs. Each server is equipped with 8 NVIDIA V100 32GB GPUs and 756GB RAM. We execute our search and experiments on Python 3 using `PyTorch==1.8.1` and `TensorFlow==1.15.0`. We implement our predictors using `PyTorch-Geometric==1.7.1`. We use SentencePiece [33] to perform BPE. Finally, we implement our MILP using a Coin-CBC solver [18] and `pyomo==6.4.0` [23].

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, number 2016, pages 265–283. Savannah, GA, USA, 2016.

[2] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[3] Kiyoharu Aizawa, Azuma Fujimoto, Atsushi Otsubo, Toru Ogawa, Yusuke Matsui, Koki Tsubota, and Hikaru Ikuta. Building a manga dataset "manga109" with annotations for multimedia applications. *IEEE MultiMedia*, 27(2):8–18, 2020.

[4] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. `https://github.com/onnx/onnx`, 2019.

[6] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14323–14332, 2020.

[7] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smaïl Niar, Martin Wistuba, and Naigang Wang. A comprehensive survey on hardware-aware neural architecture search. *CoRR*, abs/2101.09336, 2021.

[8] Marco Bevilacqua, Aline Roumy, Christine Guillemot, and Marie Line Alberi-Morel. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. 2012.

[9] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

[10] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020.

[11] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303, 2019.

[12] Krishna Teja Chitty-Venkata, Murali Emani, Venkatram Vishwanath, and Arun K Somani. Neural architecture search for transformers: A survey. *IEEE Access*, 2022.

[13] Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6639–6649, 2021.

[14] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[15] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Péter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using predictor pretraining. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 16271–16280, 2021.

[16] Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network. In *European conference on computer vision*, pages 391–407. Springer, 2016.

[17] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[18] John Forrest and Robin Lougee. *CBC User Guide*, pages 257–277. 09 2005.

[19] Philippe Fournier-Viger, Chao Cheng, Chun-Wei Lin, Unil Yun, and Rage Uday Kiran. Tkg: Efficient mining of top-k frequent subgraphs. In *BDA*, 2019.

[20] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

[21] Ehsan Goodarzi, Mina Ziaei, and Edward Zia Hosseinipour. *Introduction to optimization analysis in hydrosystem engineering*. Springer, 2014.

[22] Fred X. Han, Keith G. Mills, Fabian Chudak, Parsa Riahi, Mohammad Salameh, Jialin Zhang, Wei Lu, Shangling Jui, and Di Niu. A general-purpose transferable predictor for neural architecture search. In *Proceedings of the 2023 SIAM International Conference on Data Mining (SDM)*. SIAM, 2023.

[23] William E. Hart, Carl D. Laird, Jean-Paul Watson, and David L. Woodruff. Pyomo — optimization modeling in python. *Springer Optimization and Its Applications*, 2012.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[25] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2019.

[26] Jun Huan, Wei Wang, Jan Prins, and Jiong Yang. Spin: mining maximal frequent subgraphs from graph databases. *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.

[27] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5197–5206, 2015.

[28] Andrey Ignatov, Radu Timofte, et al. Pirm challenge on perceptual image enhancement on smartphones: report. In *European Conference on Computer Vision (ECCV) Workshops*, January 2019.

[29] Zhihao Jia, Oded Padon, James Thomas, Todd Warszawski, Matei Zaharia, and Alex Aiken. Taso: optimizing deep learning computation with automatic generation of graph substitutions. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 47–62, 2019.

[30] Chuntao Jiang, Frans Coenen, and Michele A. A. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28:75 – 105, 2012.

[31] Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, Alexander Filippov, and Evgeny Burnaev. Nas-bench-nlp: neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022.

[32] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical Report*, 2009.

[33] Taku Kudo and John Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[34] Zhuo Li, Hengyi Li, and Lin Meng. Model compression for deep neural networks: A survey. *Computers*, 12(3):60, 2023.

[35] Bee Lim, Sanghyun Son, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. Enhanced deep residual networks for single image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 136–144, 2017.

[36] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[37] Shun Lu, Yu Hu, Peihao Wang, Yan Han, Jianchao Tan, Jixiang Li, Sen Yang, and Ji Liu. Pinat: A permutation invariance augmented transformer for nas predictor. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[38] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *Advances in Neural Information Processing Systems*, 33:10547–10557, 2020.

[39] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423. IEEE, 2001.

[40] Yusuke Matsui, Kota Ito, Yuji Aramaki, Azuma Fujimoto, Toru Ogawa, Toshihiko Yamasaki, and Kiyoharu Aizawa. Sketch-based manga retrieval using manga109 dataset. *Multimedia Tools and Applications*, 76(20):21811–21838, 2017.

[41] Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *CoRR*, abs/2106.08962, 2021.

[42] Keith G Mills, Fred X Han, Mohammad Salameh, Seyed Saeed Changiz Rezaei, Linglong Kong, Wei Lu, Shuo Lian, Shangling Jui, and Di Niu. L2nas: Learning to optimize neural architectures via continuous-action reinforcement learning. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 1284–1293, 2021.

[43] Keith G. Mills, Fred X. Han, Jialin Zhang, Fabian Chudak, Ali Safari Mamaghani, Mohammad Salameh, Wei Lu, Shangling Jui, and Di Niu. Gennape: Towards generalized neural architecture performance estimators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[44] Keith G. Mills, Fred X. Han, Jialin Zhang, Seyed Saeed Changiz Rezaei, Fabián A. Chudak, Wei Lu, Shuo Lian, Shangling Jui, and Di Niu. Profiling neural blocks and design spaces for mobile neural architecture search. *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021.

[45] Keith G. Mills, Di Niu, Mohammad Salameh, Weichen Qiu, Fred X. Han, Puyuan Liu, Jialin Zhang, Wei Lu, and Shangling Jui. Aio-p: Expanding neural performance predictors beyond image classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

[46] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[47] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[48] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[49] Seyed Saeed Changiz Rezaei, Fred X Han, Di Niu, Mohammad Salameh, Keith Mills, Shuo Lian, Wei Lu, and Shangling Jui. Generative adversarial neural architecture search. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2227–2234. International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[51] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael A. Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *ICLR*, 2021.

[52] Robin Ru, Pedro Esperança, and Fabio Maria Carlucci. Neural architecture generator optimization. In *Advances in Neural Information Processing Systems*, volume 33, pages 12057–12069, 2020.

[53] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[54] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[55] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[56] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *ArXiv*, abs/1508.07909, 2015.

[57] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[58] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[59] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Péter Vajda, and Joseph Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12962–12971, 2020.

[60] Haojie Wang, Jidong Zhai, Mingyu Gao, Zixuan Ma, Shizhi Tang, Liyan Zheng, Yuanzhi Li, Kaiyuan Rong, Yuanyong Chen, and Zhihao Jia. Pet: Optimizing tensor programs with partially equivalent transformations and automated corrections. In *OSDI*, pages 37–54, 2021.

[61] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems*, 34:28454–28469, 2021.

[62] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

[63] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations*, 2020.

[64] Xifeng Yan and Jiawei Han. gspan: graph-based substructure pattern mining. *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, 2002.

[65] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *Knowledge Discovery and Data Mining*, 2003.

[66] Yichen Yang, Phitchaya Phothilimthana, Yisu Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. Equality saturation for tensor graph superoptimization. *Proceedings of Machine Learning and Systems*, 3:255–268, 2021.

[67] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114, 2019.

[68] Arber Zela, Julien Niklas Siems, Lucas Zimmer, Jovita Lukasik, Margret Keuper, and Frank Hutter. Surrogate NAS benchmarks: Going beyond the limited search spaces of tabular NAS benchmarks. In *International Conference on Learning Representations*, 2022.

[69] Roman Zeyde, Michael Elad, and Matan Protter. On single image scale-up using sparse-representations. In *International conference on curves and surfaces*, pages 711–730. Springer, 2012.

[70] Hengshuang Zhao. semseg. `https://github.com/hszhao/semseg`, 2019.

[71] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

[72] Ce Zheng, Wenhan Wu, Chen Chen, Taojiannan Yang, Sijie Zhu, Ju Shen, Nasser Kehtarnavaz, and Mubarak Shah. Deep learning-based human pose estimation: A survey, 2020.

[73] Xingyi Zhou, Qixing Huang, Xiao Sun, Xiangyang Xue, and Yichen Wei. Towards 3d human pose estimation in the wild: A weakly-supervised approach. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[74] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.